BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

89

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 11/00

A1

(11) International Publication Number: WO 95/12848

(43) International Publication Date: 11 May 1995 (11.05.95)

(21) International Application Number: PCT/US94/12567

(22) International Filing Date: 3 November 1994 (03.11.94)

(30) Priority Data:
08/148,138 3 November 1993 (03.11.93) US

(71) Applicant: EO, INC. [US/US]; 3195 Kifer Road, Santa Clara, CA 95051-0804 (US).

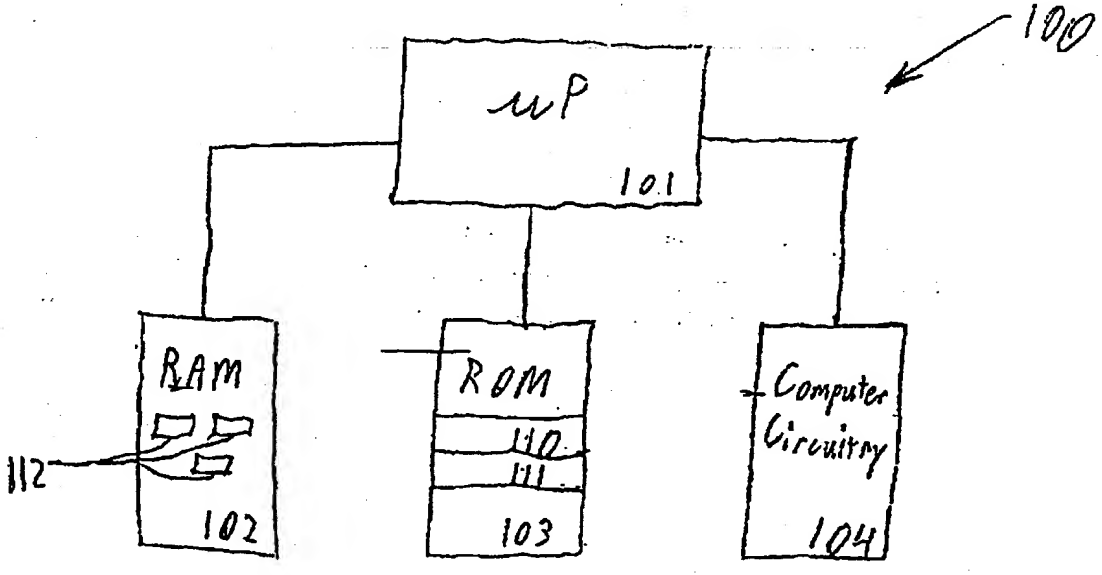
(72) Inventor: KELLOGG, Gregg, B.; 33 Bear Glenn Road, Woodside, CA 94062 (US).

(74) Agents: COLES, Anthony, C. et al.; Meltzer, Lippe, Goldstein, Wolf, Schlissel & Sazer, P.C., 190 Willis Avenue, Mineola, NY 11501 (US).

(81) Designated States: CN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: RECOVERY BOOT PROCESS



(57) Abstract

pubomputer system (100) that uses a random access memory (102) or equivalent memory to store operating system data and "persistent" data (112), i.e., any data stored in the random access memory (102) that is designated to be retained after a system reset, includes a recovery boot system that implements a method for recovering the persistent data after a system reset that necessitates re-initialization of the random access memory (102) with operating system data. The recovery boot system allocates the random access memory (102) during re-initialization such that new memory allocations do not conflict with persistent data (102) allocations made prior to the system reset. After fundamental operating system data are initialized, the persistent data (112) are recovered. Thereafter, the remainder of the data to be re-initialized can be stored in any available location in the random access memory (102).

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

RECOVERY BOOT PROCESS

Gregg B. Kellogg

CROSS-REFERENCE TO MICROFICHE APPENDICES

Appendix A, which is a part of the present disclosure, is a microfiche appendix consisting of 3 sheets of microfiche having a total of 154 frames. Microfiche Appendix A is a computer program listing of boot code for use in one embodiment of this invention, which is described more completely below.

Appendix B, which is a part of the present disclosure, is a microfiche appendix consisting of 1 sheet of microfiche having a total of 21 frames. Microfiche Appendix B is a ROM valid listing defining the data structure of a ROM according to one embodiment of this invention, which is described more completely below.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to a computer system including a memory and a method for resetting the computer system such that pre-existing persistent data stored within the computer system is not destroyed by the system reset.

2. Related Art

In computer systems, various events can cause the computer operating system to "crash," disabling the computer system. An operating system crash is initiated, for example, when the integrity of certain operating system data is compromised. Operating system data can be

compromised by, for example, an internal software error, a hardware failure, an application failure associated with an insufficiently guarded operating system, or random memory decay due to alpha-particle interaction with DRAM cells.

An operating system crash is manifested by detection and reporting of an error by the operating system, or by "freezing up" of the computer system so that no further operations can be executed on the computer system. When the operating system crashes, the computer system must be reset and some or all of the computer system hardware re-initialized to a state at which user operation of the computer system can begin again.

The system reset may be either a hardware reset or a software reset, as known to those skilled in the art. After a hardware reset, all of the computer system hardware is re-initialized. After a software reset, only the primary memory device, e.g., a random access memory (RAM), which stores the operating system data is re-initialized with initial (typically, non-zero) data values. Herein, "primary memory device" refers to a memory device from which the computer system microprocessor generally works during operation of the computer system. "Primary memory device" is distinct from a "secondary memory device" such as a disk drive or a tape unit. After both the hardware and software resets, the primary memory device must be re-initialized.

Depending on the nature of the operating system crash, the operating system itself may automatically initiate a system reset, or the user must initiate a system reset. User-initiated hardware resets are typically initiated by turning the computer system off and then on again. User-initiated software resets are typically initiated by pushing a reset button on the computer system, or one or more keys on the keyboard.

For most computer systems, a system reset is frequently a relatively benign operation that does not result in the destruction of any user data. This is because user data can be stored on one or more secondary memory devices, such as a disk drive, that are separate from the primary memory device on which the operating system is stored. Thus, even though user data is typically initially stored on the primary memory device, if the user data has been "permanently" saved to the secondary memory device, as is frequently the case, initiation of a system reset does not destroy user data since only the data stored on the primary memory device is re-initialized. Though data in the secondary memory device can also be compromised, existing recovery software allows restoration of the integrity of such data.

In contrast, computer systems containing only a primary memory device store user data along with operating system data at all times. Computer systems including only a primary memory device are desirable because the price of the computer system is reduced since a secondary memory device is not required and since tighter integration of hardware components is possible because user data is directly available.

However, a significant problem with computer systems including only a primary memory device is that a system reset typically causes all user data to be lost when the operating system data is re-initialized on the primary memory device. In existing methods for re-initializing a primary memory device after a system reset, prior to re-initialization, all of the primary memory locations are placed on a "free list." The operating system data is then stored at any available primary memory location. Thus, the operating system data may be stored in a primary memory location that previously contained user data, resulting in the destruction of the user data.

Thus, it is desirable to provide, in a computer system including only a primary memory device, a system and method for re-initializing the operating system on the primary memory device after a system reset, such that user data existing prior to the system reset is preserved after the system reset.

SUMMARY OF THE INVENTION

In a computer system that uses a random access memory or equivalent memory to store operating system data and persistent data, i.e., any data stored in the memory that is designated to be retained after a system reset, a recovery boot system according to the invention implements a method for recovering the persistent data after a system reset that necessitates re-initialization of operating system data into the memory. Persistent data as well as other data stored within the memory is not destroyed by the system reset. The recovery boot system according to the invention allocates memory during re-initialization of the operating system data into the memory such that new memory allocations do not conflict with persistent data allocations made prior to the system reset, and stores data within the memory such that the data is not stored in locations that contain persistent data.

In the recovery boot system according to the invention, memory allocations are made and fundamental operating system data is stored in the memory to allow the operating system kernel and hardware interface code to begin running. Memory allocations are made to recover persistent data. Each allocated memory location is marked to indicate that the memory location is no longer available to store data. Using only the remaining, unmarked memory locations, the remainder of the operating system is initialized.

Each instance of data stored in the memory is associated with a virtual memory address used by a

processor (also part of the computer system) to identify the physical memory address of the data and access the physical memory location of the data in the memory. Data required for translating the virtual memory addresses of both operating system data and persistent data to physical memory addresses are retained, in one embodiment, through the system reset.

After the system reset, the retained translation data are used to map the virtual memory addresses of fundamental operating system data to physical memory addresses. Consequently, after the system reset, fundamental operating system data is stored in the same physical memory locations in the memory as before the system reset. Thus, re-initialization of the fundamental operating system data in the memory does not destroy any of the persistent data stored in the memory.

After memory allocation for the fundamental operating system data, i.e., recovery of the fundamental operating system data, the persistent data is recovered. During normal operation of the computer system, the virtual memory addresses of some data stored in the memory are tagged to indicate that the data is persistent data. A list of tagged virtual memory addresses is stored starting at a physical memory address corresponding to a virtual memory address that is fixed for the operating system. After system reset, this virtual memory address, which can be "known" by the processor immediately after the system reset or "discovered" by the processor during recovery of the fundamental operating system data, is used to access the list of tagged virtual memory addresses. The tagged virtual memory addresses are used to allocate the corresponding physical memory locations in the memory so that other data cannot thereafter be stored in those memory locations during the recovery boot process.

Subsequently, the recovery boot process can be completed using any physical memory locations that have

not previously been allocated. Since only physical memory locations are used that have not been allocated during the recovery of the fundamental operating system data and the persistent data, persistent data is not destroyed by the remainder of the recovery boot process.

An advantage of the system and method according to the invention is that persistent data can be recovered after a system reset without unduly constraining the usage of memory. For example, it is not necessary, as is the case of other systems, to specify a region of memory for storage of the file system. Unlike other systems in which a pre-defined area is set aside for file system storage, i.e., a region in memory of fixed size is allocated only for the file system, in the recovery boot system and method according to the invention, the exact locations and sizes of all portions of the file system are determined at boot time. Thus, memory fragmentation, i.e., memory space allocated for a specific type of data that remains unused by data of that type, does not occur with the recovery boot system and method of the invention. Consequently, the recovery boot system and method according to the invention allow a more dynamic allocation of memory between file system data and run time data. Such flexibility is particularly useful for computer systems that include an operating system with complex file system structures that require a large area of memory for run time data storage, since more memory is made available for those run time data structures.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating a computer system according to the invention.

Figure 2A is a block diagram of a recovery boot process according to an embodiment of the invention.

Figure 2B is a block diagram of a recovery boot process according to another embodiment of the invention.

Figure 3A is a schematic representation of mapping of a virtual memory address to a physical memory location using paged segmentation according to one embodiment of the invention.

Figure 3B is a schematic representation of mapping of a virtual memory address to a physical memory location using non-paged segmentation according to another embodiment of the invention.

Figure 4A is a diagram of the data structure of a ROM for use with the invention.

Figure 4B is a diagram of a ROM item defined within the ROM of Figure 4A.

Figure 4C is a diagram of an Elf file defined within the ROM item of Figure 4B.

Figure 5A is a block diagram of the fundamental operating system data initialization step of Figure 2A according to an embodiment of the invention.

Figure 5B is a block diagram of the fundamental operating system data initialization step of Figure 2A according to another embodiment of the invention.

Figure 6 is a block diagram of the persistent region recovery step of Figure 2A according to an embodiment of the invention.

Figure 7A is a block diagram of the system free list initialization step of Figure 2A according to an embodiment of the invention.

Figure 7B is a block diagram of the system free list initialization step of Figure 2B according to an embodiment of the invention.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

According to the principles of the invention, a computer system that uses a random access memory or equivalent memory to store operating system data and "persistent" data (defined below) includes a recovery boot system that implements a method for recovering the

persistent data after a system reset that necessitates re-initialization of operating system data into the memory. Specifically, the recovery boot system according to the invention allocates memory during re-initialization of the operating system data into the memory such that new memory allocations do not conflict with persistent data allocations made prior to the system reset.

Herein, "persistent data" is any data stored in the memory that is designated to be retained after a system reset. Persistent data typically includes data that is generated during use of the computer system that would take a significant amount of effort on the part of a user of the computer system to recreate, or that probably cannot be exactly recreated by the user. In one embodiment, persistent data includes, for instance, the file system data created and continuously updated during use of the computer system. The file system data includes, for example, data generated by a user such as text created using a word processing application program or a database created using a database application program.

Each instance of data stored in the memory is associated with a virtual memory address used by a processor (also part of the computer system) to identify and access the physical memory location of the data in the memory. Data required for translating the virtual memory addresses of both operating system data and persistent data to physical memory addresses are retained through the system reset. After the system reset, the retained translation data are used to map the virtual memory addresses of fundamental operating system data to physical memory addresses. The fundamental operating system data is data that is sufficient to allow operation of the kernel and code that interfaces the operating system to various system hardware. Consequently, after the system reset, the fundamental operating system data is stored in

the same physical memory locations as before the system reset. Since these physical memory locations are the same as the physical memory locations in which the fundamental operating system data was stored prior to the system reset, re-initialization of the fundamental operating system data does not destroy any of the persistent data. The physical memory addresses corresponding to the physical memory locations at which the fundamental operating system data are stored are marked to indicate that those physical memory locations cannot be used to store other data.

In the recovery boot process of this invention, after memory allocation for the fundamental operating system data, i.e., recovery of the fundamental operating system data, the persistent data is recovered. During normal operation of the computer system, the virtual memory addresses of some data stored in the memory are tagged to indicate that the data is persistent data. A list of tagged virtual memory addresses is stored starting at a physical memory address corresponding to a virtual memory address that is fixed for the operating system. After system reset, this virtual memory address, which can be "known" by the processor immediately after reset or "discovered" by the processor during recovery of the fundamental operating system data, as explained in more detail below, is used to access the list of tagged virtual memory addresses. The tagged virtual memory addresses are mapped to physical memory addresses and the physical memory addresses are marked to indicate that data cannot be stored at the corresponding physical memory locations.

Figure 1 is a block diagram illustrating a computer system 100 according to the invention. Microprocessor 101 interfaces with a random access memory 102, e.g., a dynamic random access memory, a read only memory (ROM) 103, and other computer circuitry 104.

In Figure 1, only the components of computer system 100 required to understand the recovery boot system and method according to this invention are illustrated. Those skilled in the art will appreciate that computer system 100 also includes structure for accepting user input to computer system 100 such as a keyboard or a pen stylus, an output display device such as a back-lighted liquid crystal display, and structure for housing and/or supporting each of the components of computer system 100. Additionally, each of the components of computer system 100 are interconnected using structure known to those skilled in the art. A computer system 100 suitable for inclusion of the recovery boot system according to the invention is sold by EO Inc. of Mountain View, California as Model No. 440 or Model No. 880.

ROM 103 has a data structure discussed more completely below, and stores all of the operating system data 111, i.e., basic instructions for operating microprocessor 101, for computer system 100, such as kernel instructions, instructions for various libraries and services, and instructions for graphics libraries, as well as instructions 112 for the novel recovery boot system according to the invention. Any operating system compatible with microprocessor 101 can be used. In one embodiment, where microprocessor 101 is a Hobbit processor available from American Telephone and Telegraph (AT&T), the operating system is the PenPoint operating system available from Go Corp. ROM 103 also stores applications programs such as word processing, data communications and graphics programs.

RAM 102 is used during operation of computer system 100 to store data that it is desirable to make readily accessible to microprocessor 101, e.g., data that is frequently used, or read/write data such as user data. Some of the user data stored on RAM 102 is persistent data 112, as defined above.

During operation, computer system 100 may "crash" for any of a number of reasons such as, for instance, an internal software error, i.e., corruption of some of the data stored on RAM 102. Often, the operating system is able to recover from the crash on its own and does not require resetting of computer system 100. A crash from which the operating system cannot recover may be indicated, for instance, by detection and reporting of an error or by the failure of computer system 100 to respond to user input. When the operating system cannot recover on its own, computer system 100 must be reset and RAM 102 must be re-initialized with operating system data 111 to a state that allows user operation of computer system 100. Computer system 100 contains hardware that allows the user to "manually" reset computer system 100 after a crash from which the operating system cannot automatically recover.

Computer system 100 can be manually reset in two ways: hardware reset and software reset (recovery boot). A hardware reset causes computer circuitry 104, RAM 102 and certain other hardware, not shown for clarity in Figure 1, to be re-initialized. In particular, RAM 102 is re-initialized without regard to the state of RAM 102 prior to the hardware reset. Hence, all data stored in RAM 102 prior to the hardware reset, including persistent data 112, is lost. A software reset is less drastic. RAM 102 is re-initialized, as described below, so as to prevent certain important data, i.e., persistent data 112, from being lost.

Generally, a software reset is preferred. A hardware reset is necessitated if the software reset does not return computer system 100 to a user operable state. A hardware reset may also be required, even though the software reset "works," i.e., computer system 100 can be re-initialized to a user operable state, if, after the software reset, the persistent data 112 is corrupted to the point that the persistent data 112 is not usable.

In one embodiment of the invention, a hardware reset is initiated by depressing a reset button so that, while power remains turned on to computer system 100, computer circuitry 104 and RAM 102 are fully re-initialized.

In one embodiment of the invention, a software reset is initiated by depressing the power button for an extended period of time, then releasing the power button. Any desired length of time can be used and the passage of sufficient time can be indicated by, for instance, an audio signal such as a succession of beeps. In a further embodiment, the tone of a succession of beeps changes when the power button has been depressed for a sufficient time to initiate a software reset. Depression and release of the power button as described above each generate an interrupt so that a software reset is initiated.

Figure 2A is a block diagram of recovery boot process 200 according to an embodiment of this invention. Recovery boot process 200 begins with system reset step 205. System reset step 205 includes a user-initiated software reset, as described above. Immediately after initiation of the software reset, certain hardware, such as a power controller, a communications port, and a display screen, are initialized. One embodiment of a power controller suitable for use with this invention is described in copending, commonly owned, U.S. Patent Application Serial No. ??/???,??? entitled "A Power Supply Circuit for Powering a Portable Computer and an External Device from a Single Battery Power Source," by David Anthony Chavez, Ron A. Balczewski, and Bernard Jean Lacroute, filed on the same date as the present application, the pertinent disclosure of which is herein incorporated by reference.

RAM 102 is initially unavailable for use by microprocessor 101 during system reset step 205 so that there is no danger of destruction of persistent data 112 stored in RAM 102. As explained in more detail below, as

recovery boot process 200 progresses through the initial stages, certain pre-defined locations of RAM 102 gradually become available for use by microprocessor 101. Microprocessor 101 does not have free use of RAM 102 until persistent data 112 has been recovered, as explained below.

In fundamental operating system data initialization step 210, certain operating system data, referred to herein as "fundamental operating system data," that are necessary to enable operation of the kernel and the mill are recovered to permit a basic level of operation of computer system 100. The fundamental operating system data are stored in ROM 103.

In one embodiment, at the beginning of fundamental operating system data initialization step 210, only a physical memory address in ROM 103 of a data structure that enables boot code, i.e., code that controls the initialization of the hardware of computer system 100, to begin to be processed by microprocessor 101 is available to microprocessor 101, as described in more detail below. The boot code includes instructions that, along with the known ROM data structure, allow "discovery" of physical memory locations in ROM 103 that store virtual memory addresses and physical memory addresses of the fundamental operating system data as existent before system reset step 205. These virtual and physical memory addresses are used to re-initialize segment and page table entries corresponding to the fundamental operating system data, and to enable copying of certain fundamental operating system data to RAM 102. Thus, the fundamental operating system data is stored at the same physical memory locations as were used before recovery boot process 200, thereby ensuring that recovery of the fundamental operating system data does not destroy persistent data 112.

According to the invention, recovery boot process 200 recovers persistent data 112 stored in RAM 102 so that persistent data 112 is not destroyed by the re-initialization of RAM 102 with operating system data. Persistent data 112 was marked as persistent at some time before system reset step 205. For example, persistent data 112 can be marked as persistent during the first initialization of the operating system of computer system 100, i.e., when computer system 100 is first turned on, after a hardware reset, or during normal operation of computer system 100. In one embodiment of the invention, data representing the file system of computer system 100 is marked as persistent data 112. The file system data is initially created during the first initialization of the operating system of computer system 100 and updated during normal operation of computer system 100.

After completion of fundamental operating system data initialization step 210, persistent data 112 is recovered in persistent region recovery step 220. A region table, the virtual memory address of which is pre-defined by the operating system and "discovered" during fundamental operating system data initialization step 210, includes a list of region table entries, each region table entry defining a region of virtual memory space. Each region table entry includes the range of virtual memory addresses included within the region and an attribute that indicates whether or not the virtual memory addresses within the region correspond to physical memory locations that store persistent data 112. The region table is accessed and reviewed. For each virtual memory address of each region that stores persistent data 112, the corresponding segment table and page table entries are marked to indicate that corresponding physical memory locations are allocated.

After completion of persistent region recovery step 220, a system free list is created in system free list initialization step 230. For each region in the

region table, the segment and page table entries corresponding to virtual addresses in the region are reviewed. Physical memory addresses of physical memory locations that are being used are identified and the system free list is created as a list of the remaining physical memory addresses.

In contrast, in previous methods for re-initializing RAM after a software reset, all of the RAM memory locations were immediately placed on the system free list. The operating system data was mapped to any available physical memory locations within the RAM. Thus, the operating system data could be stored in physical memory locations that contain user data prior to the system reset, thereby destroying the user data.

After system free list initialization step 230 is completed, the next step in recovery boot process 200 is system start-up step 240. System start-up step 240 includes, among other things, initialization of input/output devices, further initialization of parts of the operating system kernel, e.g., memory manager and memory allocator, and initialization of device drivers.

After system start-up step 240, the next step in recovery boot process 200 is file system recovery step 250. In file system recovery step 250, the data structure, i.e., file system, that stores information regarding the files stored in RAM 102 and ROM 103 is recovered. The file system includes information regarding the name, location, permission and attributes of each file. The file system also includes information regarding the presence of other volumes or disks.

Upon completion of file system recovery step 250, computer system 100 completes recovery boot process 200 by initializing the remaining parts of the operating system such as additional libraries and applications that are part of the operating system.

Figure 2B is a block diagram of recovery boot process 260 according to another embodiment of the invention. Recovery boot process is similar to recovery boot process 200 and like steps are indicated by the same numerals used in Figure 2A. The above description of steps in process 200 with the same reference numeral as the steps in process 260 are incorporated herein by reference.

In recovery boot process 260, fundamental operating system data initialization step 211 and system free list initialization step 231 are slightly different than fundamental operating system data initialization step 210 and system free list initialization step 230, respectively, and persistent data recovery 220 is eliminated. Recovery boot process 260 differs from recovery boot process 200 in that fundamental operating system data and persistent data 112 that are recovered are not marked as used during recovery boot process 260, i.e., the segment and page table entries corresponding to fundamental operating system data and persistent data 112 are not marked when recovered. Rather, recovery boot process 260 takes advantage of the fact that during operation of computer system 100 prior to system reset step 205, both fundamental operating system data and persistent data 112 are marked in the region table as "persistent," i.e., a persistent attribute is "turned on." During system free list initialization 231, the region table is scanned and the physical memory addresses of all persistent data 112 are removed from the system free list. Virtual addresses of persistent data 112 are also removed from the list of available virtual memory.

Above, recovery boot processes 200 and 260 included a system reset step 205 that, in turn, included a software reset. Computer system 100 also can recover from a processor reset in a manner similar to recovery boot process 200 or 260. A processor reset is a non-user-

initiated reset, i.e., an automatic reset, that occurs when microprocessor 101 attempts to take an exception, and the process of taking an exception takes an exception. At that point, the only thing microprocessor 101 can do is reset. In a method according to the invention for recovering after a processor reset, microprocessor 101 goes through a processor reset state in which microprocessor 101 determines whether a hardware or software reset is required, and, if a software reset is required, begins one of recovery boot process 200 or 260.

In computer system 100, the segment table and, therefore, page tables are available to microprocessor 101 shortly after system reset step 205 because the physical memory base address of the segment table is "discovered," as explained in more detail below, during the initial stages of fundamental operating system data initialization step 210. Discovery of this physical memory base address can be accomplished either by structuring the operating system to store the physical memory base address of the segment table, e.g., at a physical memory address identified within the boot data structure, the boot data structure having been "discovered" at an earlier stage of fundamental operating system data initialization step 210, or by structuring the computer system hardware so that the physical memory base address of the segment table is always known, e.g., by having one of the registers of microprocessor 101 store the physical memory base address of the segment table.

Recovery boot process 200 takes advantage of the preservation of the physical memory base address of the segment table to, with other tables, re-initialize RAM 102, as explained below, without destroying persistent data 112 stored in RAM 102 prior to the system reset. The segment table and associated page tables are used to translate virtual memory addresses into physical memory addresses of persistent data 112 so that the persistent

data 112 is restored to the same physical memory locations in RAM 102 as used prior to recovery boot process 200 or 260.

Computer system 100 has a memory structure including a physical memory address space and a virtual memory address space. The physical memory address space includes the entire set of physical memory addresses. The system controller (not shown) associates portions of the physical memory address space with particular pieces of hardware, i.e. one or more particular physical memory locations, such as ROM 103, RAM 102 or input/output ports (not shown). However, not all physical memory addresses necessarily refer to actual physical memory locations, and more than one physical memory address can refer to the same physical memory location. The virtual memory address space includes the entire set of virtual memory addresses. Portions of the virtual memory address space are associated with particular pieces of software, either dynamically or statically.

In one embodiment, the virtual memory address space and the physical memory address space are the same size. The software of computer system 100 accesses data stored within the hardware of computer system 100 by translating known virtual memory addresses to physical memory addresses and accessing the data stored at the corresponding physical memory location.

Figure 3A is a schematic representation of mapping of virtual memory address 300 to physical memory location 331-3 in physical memory page 330 using paged segmentation according to one embodiment of the invention. Virtual memory address 300 is four bytes (32 bits) long and includes segment table offset 301, page table offset 302 and page offset 303. Segment table offset 301 includes 10 bits (bits 22 through 31), page table offset 302 includes 10 bits (bits 11 through 21), and page offset 303 includes 12 bits (bits 0 through 11). It is to

be understood that virtual memory address 300 could be one, two, eight or some other number of bytes long, and that segment table offset 301, page table offset 302 and page offset 303 can include any desired number of bits.

Segment table 310, page table 320 and memory page 330 are used by computer system 100 in translation of virtual memory address 300 to physical memory location 331-3. Each of segment table 310 and page table 320 are themselves a memory page. Segment table 310, page table 320 and memory page 330 each include 1000 four byte words. In segment table 310, each word is one of the segment table entries 311-1 through 311-N. In page table 320, each word is one of the page table entries 321-1 through 321-N. In memory page 330, each word is one of the physical memory locations 331-1 through 331-N.

Herein, "word" refers to the primary unit of information used by microprocessor 101. It is to be understood that the computer system and method according to the invention are equally applicable to computer systems having a segment table, page tables and memory pages including a different number of words or words of a different size.

Segment table entry 311-2 includes physical memory address 312, and region 313 that includes permission and read/write attributes. Each of the other segment table entries 311-1 and 311-3 through 311-N similarly include a physical memory location, and permission and read/write attributes. Likewise, page table entry 321-4 includes physical memory address 322, and a region 323 that includes permission and read/write attributes. Each of the other page table entries 321-1 through 311-3 and 321-5 through 321-N similarly include a physical memory location, and permission and read/write attributes. The read/write attributes in regions 313 and 323 indicate whether the segment table entry 311-2 and page table entry 321-4, respectively, have been written to or read

from. The permission attributes in regions 313 and 323 indicate whether the segment table entry 311-2 and page table entry 321-4, respectively, are protected at kernel level or user level.

Though not shown, there are a plurality of page tables, e.g., page table 320. The physical memory address in each segment table entry, e.g., physical memory address 312 in segment table entry 311-2, is a physical memory base address of one of the page tables, e.g., physical memory base address 324 of page table 320.

Likewise, there are a plurality of memory pages, e.g., memory page 330. The physical memory address in each page table entry, e.g., physical memory address 322 in page table entry 321-4, is a physical memory base address of one of the memory pages, e.g., physical memory base address 334 of memory page 330.

Translation of virtual memory address 300 to physical memory location 331-3 occurs as follows. Segment table offset 301 in virtual memory address 300 is combined with physical memory base address 314 of segment table 310 to yield physical memory address 315 of segment table entry 311-2. Physical memory address 312 in segment table entry 311-2 specifies the physical memory base address 324 of page table 320.

Page table offset 302 in virtual memory address 300 is combined with physical memory base address 324 to yield physical memory address 325 of page table entry 321-4. Physical memory address 322 in page table entry 321-4 specifies physical memory base address 334 of memory page 330. Page offset 303 in virtual memory address 300 is combined with physical memory base address 334 to yield physical memory address 335 of physical memory location 331-3.

Figure 3B is a schematic representation of mapping of virtual memory address 340 to physical memory location 361-2 using non-paged segmentation according to

another embodiment of the invention. Virtual memory address 340 is four bytes long and includes segment table offset 341 and segment offset 342. Segment table offset 341 includes 10 bits (bits 22 through 31) and segment offset 342 includes 22 bits (bits 0 through 21). As above, virtual memory address 340 can be other than four bytes long, and segment table offset 341 and segment offset 341 can include any number of bits.

Translation of virtual memory address 340 to physical memory location 361-2 occurs as follows. Segment table offset 341 in virtual memory address 340 is combined with physical memory base address 314 of segment table 310 to yield physical memory address 355 of segment table entry 311-4. Physical memory address 352 in segment table entry 311-4 specifies physical memory base address 364 of segment 360. Segment offset 342 in virtual memory address 340 is combined with physical memory base address 364 to yield physical memory address 365 of physical memory location 361-2.

Each segment table entry 311-1 through 311-N includes in region 313 or 353 a bit which indicates whether the physical memory address, e.g., physical memory address 312 or 352, in segment table entry 311-1 through 311-N points to a page table (paged segmentation) or to a segment (non-paged segmentation). In one embodiment, only paged segmentation is used by recovery boot process 200 for translation of virtual memory addresses to physical memory locations.

As described above, a portion of RAM 102 must be initialized with fundamental operating system data from ROM 103 in fundamental operating system data initialization step 210 before persistent data 112 can be recovered in persistent region recovery step 220. To begin fundamental operating system data initialization step 210, microprocessor 101 accesses a predetermined memory location in ROM 103 which, in turn, enables access

to a memory location or locations in ROM 103 which store mapping information, i.e., information matching physical memory addresses in ROM 103 to virtual memory addresses, for the data stored on ROM 103. This mapping information is used to re-initialize segment table and page table entries for fundamental operating system data stored on ROM 103.

In one embodiment, each re-initialized segment table and page table entry is marked to indicate that the particular physical page or pages corresponding to that entry are no longer available, i.e., future allocations of data cannot be made to these physical memory locations. Thus, the mappings of fundamental operating system data that existed before the system reset are preserved after the system reset through use of the information in ROM 103. Consequently, no fundamental operating system data is mapped to a physical memory location that contains persistent data 112 so that no persistent data 112 can be destroyed by the fundamental operating system data initialization step 210. Persistent data 112 can then be recovered during persistent data recovery step 220. Subsequent to persistent data recovery step 220, the remainder of recovery boot process 200 can proceed without regard to the particular mappings of data to physical memory locations.

Figure 4A is a diagram of the data structure of ROM 410. ROM 410 includes ROM data header 411, map table 412, and ROM items 413-1 through 413-N. Hereafter, ROM items 413-1 through 413-N are designated collectively by the numeral 413; particular ROM items are designated as one of ROM items 413-1 through 413-N, e.g., ROM item 413-1. ROM data header 411 includes, among other things, the beginning physical memory address of map table 412 in ROM 103, the number of entries in map table 412, the beginning physical memory address of the first ROM item 413-1 and the number of ROM items 413.

Figure 5A is a block diagram of fundamental operating system data initialization 210 according to an embodiment of the invention. In system reset step 205, a program counter of microprocessor 101 is set to zero so that microprocessor 101 knows to access the beginning data (for the Hobbit microprocessor, the first six bytes) in ROM data header 411. The physical memory address of ROM data header 411 is hard-wired into microprocessor 101 so that microprocessor 101 always knows where to locate ROM data header 411.

As shown in step 501, microprocessor 101 locates and accesses ROM data header 411. Microprocessor 101 is instructed to jump to a ROM physical memory address which marks the beginning of boot code for controlling recovery boot process 200. Appendix A is a computer program listing of boot code for use in one embodiment of this invention. The boot code is written in the C computer language and Hobbit assembly language for an AT&T 92000 family set of computer chips including a Hobbit processor and ASIC chip set. The boot code is executed by microprocessor 101 by directly accessing the physical memory addresses of the boot code, i.e., translation of virtual addresses using the segment table and page tables is not necessary.

Among other things, the boot code looks at a register of microprocessor 101 to determine if system reset step 205 is a hardware or software reset. As noted above, recovery boot process 200 takes place only if a software reset has been initiated. If a software reset has been initiated, this microprocessor register indicates the physical memory base address of the segment table, an important piece of information that is used later in recovery boot process 200.

Near the end of execution of the boot code, the boot code instructs microprocessor 101 to access ROM data header 411 again. Microprocessor 101 retrieves the

physical memory base address of map table 412 and accesses map table 412, as shown in step 502. Map table 412 includes a series of map table entries. Each map table entry includes: (i) information indicating the type of data, i.e., executable image of code, read-only data or read/write data, referred to by the map table entry, (ii) permission bits for the data, e.g., kernel or user access allowed, (iii) instructions to either copy the data from ROM 103 to RAM 102 and map the data from RAM 102 to virtual addresses, or map the data directly from ROM 103 to virtual addresses, (iv) a virtual memory address offset and a virtual memory address length, and (v) a ROM offset and a ROM address length.

In step 503, microprocessor 101 accesses the first map table entry. In step 504, a determination is made as to whether a ROM mapping or a RAM mapping is to be made.

In one embodiment of the invention, if a ROM mapping is to be made, microprocessor 101 uses each of one or more map table entries to map all of the data stored in ROM 103 to a corresponding set of virtual addresses. In another embodiment of the invention, microprocessor 101 uses each of one or more map table entries to map data from one or more portions of ROM 103. For example, one or more map table entries can be used to map fundamental operating system data from ROM 103.

In step 507, the virtual memory address offset and virtual memory address length in the map table entry are used to determine a range of virtual memory addresses for the data described by the map table entry. The ROM offset and ROM address length in the map table entry are used to determine a range of physical memory addresses in ROM 103 of the data described by the map table entry. Using the determined virtual memory address range and physical memory address range, the segment table and page table entries corresponding to these data are initialized. This is done even though, typically, each of the segment table

and page table entries that map ROM data from the virtual memory address space to the physical memory address space already exist. Re-initialization ensures that any corrupted entries in the segment table and/or page tables are corrected, and allows the opportunity to reset permission bits, e.g., data protected at kernel or user level, in the segment table and/or page table entries.

If, in step 504, it is determined that a RAM mapping is to be made, i.e., sections of data stored in ROM 103 are to be copied to RAM 102, in step 505, microprocessor 101 accesses the physical memory location in ROM 103 corresponding to the beginning physical memory address given by the ROM offset. The ROM address length indicates the number of physical memory addresses, i.e., amount of data, from which data are to be copied to RAM 102. The virtual memory address offset and virtual memory address length in the map table entry are used to determine a range of virtual memory addresses corresponding to the data to be copied from ROM 103. The amount of data copied from ROM 103 can be equal to or less than the magnitude of the virtual memory address range. The data to be copied can be either an executable image of code, read-only data or read/write data.

As shown in step 506, data is copied from ROM 103 to RAM 102 and mapped from RAM 102 to virtual addresses. The segment table and page tables in RAM 102 are used to determine corresponding physical memory addresses, i.e., physical memory locations in RAM 102, for each of the virtual memory addresses. At this point, the mapping and copying can be done in any order. Moving sequentially through the data in ROM 103, microprocessor 101 copies each piece of data to a physical memory location corresponding to the physical memory address translated from the virtual memory address that corresponds to the piece of data being copied.

The data in RAM 102 are also mapped to virtual addresses, i.e., the corresponding segment and page table entries are initialized. The physical addresses in RAM 102 are used, along with the corresponding virtual addresses, to initialize the segment table and page tables. This is done, even though each of the segment table and page table entries must already exist, so that permission bits can be reset for each of the segment table and page table entries.

Since the segment table and page table entries used to determine the physical memory addresses in RAM 102 at which to copy data from ROM 103 are the same as the entries existing prior to system reset step 205, the data in ROM 103 is stored in the same physical memory locations in RAM 102 as before the system reset, so that no persistent data 112 are inadvertently destroyed. Unlike the direct mapping of data from ROM 103 (step 507), copying of data from ROM 103 to RAM 102 necessitates that segment table and page table entries for the data be uncorrupted.

The physical memory pages of RAM 102 to which fundamental operating system data has been copied are removed from the system free list during system free list initialization step 230 by checking a region table, described in more detail below, and removing the pages after determining that the pages are marked as persistent in the region table, as also described in more detail below.

According to one embodiment of the invention, in step 510, during mapping of data from RAM 102 to virtual addresses, a bit in each of the segment table and page table entries is marked to indicate that the corresponding physical memory location has been used. The marked entries are then eliminated from the system free list during system free list initialization step 230, discussed in detail below. This embodiment of fundamental operating

system data initialization step 210 is used in recovery boot process 200 illustrated in Figure 2A.

According to another embodiment of the invention, the segment table and page table entries are not marked, i.e., step 510 is eliminated from the method. The physical memory pages of RAM 102 to which fundamental operating system data has been copied are removed from the system free list during system free list initialization step 231 (Figure 2B) by checking a region table, described in more detail below, and removing the pages after determining that the pages are marked as persistent in the region table, as also described in more detail below. This embodiment is designated as fundamental operating system data initialization step 211 in recovery boot process 260 of illustrated in Figure 2B.

As shown in step 508, a determination is made as to whether the last map table entry has been processed. If not, the next map table entry is acessed (step 503) and either a ROM mapping or RAM mapping (block 504) is carried out by microprocessor 101 as described above. If the last map table entry has been accessed, then the recovery boot process continues, as shown in step 508.

Figure 5B is a block diagram of the fundamental operating system data initialization step 210 according to another embodiment of the invention. The data structure of ROM 103 is the same for each of the embodiments of fundamental operating system data initialization step 210, shown in Figures 5A and 5B, respectively.

As in the embodiment of step 210 illustrated in Figure 5A, in step 501, ROM data header 411 in ROM 103 is located and accessed. Boot code execution is initiated and, during execution of the boot code, ROM data header 411 is accessed again to determine the location of map table 412.

In step 502, map table 512 is located and accessed. Map table 412 includes map table entries, each map table

entry including information as discussed above with respect to Figure 5A.

In step 511, the map table entries are used to map data as described above with respect to steps 505, 506 and 507 of Figure 5A. After all of the map table entries have been used to map data, the boot code instructs microprocessor 101 to access ROM data header 411. Microprocessor 101 accesses ROM data header 411 to determine the beginning physical memory address of the first ROM item 413-1.

In step 512, microprocessor 101 accesses the beginning physical memory location of ROM item 413-1. If data from ROM item 413-1 is to be initialized, based upon a determination described in more detail below and shown in step 513, microprocessor 101 initializes all data from first ROM item 413-1, as also described in more detail below and shown in steps 514 and 515. Whether or not the data from first ROM item 413-1 was initialized, microprocessor 101 then moves to the physical location in ROM 103 at which the second ROM item 413-2 begins. The location of the start of the second ROM item 413-2 is known because second ROM item 413-2 begins at the physical memory address in ROM 103 immediately after the last physical memory address of ROM item 413-1. If indicated, microprocessor 101 initializes all data from second ROM item 413-2. Microprocessor 101 then continues to successively initialize data from ROM items 413, as appropriate, as described above, until the last ROM item 413-N is checked, as shown in step 517. When the last ROM item 413-N has been checked, the recovery boot process continues, as shown in step 518.

To assist in understanding steps 514 and 515 of this embodiment of fundamental operating system initialization step 210, the structure of ROM items 413 is first explained. Figure 4B is a diagram of a ROM item 413-1. Each of ROM items 413 has the same structure as described

below for ROM item 413-1. In one embodiment of the invention, ROM item 413-1 includes ROM item header 414, ROM item namespace 415, ROM item attributes 416 and ROM item data 417. ROM item header 414 includes a ROM item namespace offset, a ROM item namespace size, a ROM item attributes offset, a ROM item attributes size, a ROM item data offset, a ROM item data size, and several flags. Each of the offsets specify a beginning physical memory address in ROM 103 of the corresponding part of ROM item 413-1. Each of the sizes define the ending physical memory address in ROM 103 of the corresponding part of ROM item 413-1.

In another embodiment of the invention, the ROM item header 414 of each ROM item 413 includes the size of the ROM item 413. However, this information is not necessary to implement the invention.

The flags in ROM item header 414 indicate permission attributes of ROM item 413, i.e., whether ROM item 413 is part of the operating system kernel or a user item, and whether ROM item 413 must be initialized at boot time. This latter attribute is the most important for purposes of recovery boot process 200 because it indicates whether ROM item 413 is to be initialized during fundamental operating system data initialization step 210.

ROM item namespace 415 stores the name of ROM item 413-1. The name of the ROM item is the path name that a user would use to specify the location of the program, i.e., ROM item data 417, that is stored in ROM item 413-1. This path name enables the program stored in ROM item 413-1 to be properly located within the file system of computer system 100 during file system recovery step 250, discussed in more detail below.

ROM item attributes 416 include information about characteristics of ROM item 413 such as whether ROM item 413 is initialized at boot time, whether ROM item 413 is a kernel or user item, whether ROM item 413 is a

machine specific item, and whether ROM item 413 is a file or not.

ROM item data 417 includes, among other things described in more detail below, the data constituting the program stored in ROM item 413. ROM item data 417 is also known as an Elf file. The structure of Elf file 417 is defined in Unix™ System V, Release 4 Programmer's Guide: ANSI C and Programming Support Tools, Chapter 13.

During fundamental operating system data initialization step 210, for each ROM item 413, the appropriate flag in ROM item header 414 is checked in step 513 (Figure 5B) to see if ROM item 413 requires initialization. If ROM item 413 does not require initialization, the ROM item data offset and ROM item data size are used to determine the ending location of ROM item 413 in ROM 103 and, therefore, the beginning location of the next ROM item 413 in ROM 103, so that the next ROM item 413 can be checked to determine whether that ROM item 413 is to be initialized. If ROM item 413 requires initialization, Elf file 417, i.e., ROM item data, is examined to proceed with fundamental operating system data initialization step 210.

Figure 4C is a diagram of Elf file 417. Elf file 417 includes Elf file header 418, program header table 419, executable image of code 420, read-only data 421 and read/write data 422. Elf file header 418 includes, among other things, a program header table offset, which indicates the beginning physical memory address in ROM 103 of program header table 419, and the number of program headers in program header table 419. Program header table 419 includes a program header for each section of data ("program") in Elf file 417, i.e., executable image of code 420, read-only data 421 and read/write data 422. Read/write data 422 must be copied to RAM 102. Executable image of code 420 and read-only data 421 may be copied to RAM 102. Though, in Figure 4C, only one section of

executable image of code 420, read-only data 421 and read/write data 422 are shown in Elf file 417, it is to be understood that an Elf file 417 can include more than one section of executable image of code 420, read-only data 421 and/or read/write data 422.

Each program header includes, among other things, a flag that indicates whether the program is executable image of code 420, read-only data 421 or read/write data 422. Each program header also includes: (i) a program offset that indicates the beginning physical memory address in ROM 103 of the program, and (ii) a program size that, together with the program offset, defines the ending physical memory address in ROM 103 of the program. The program offset and size define the physical memory location of the program in ROM 103. Each program header further includes a beginning virtual memory address and a virtual program size that defines an ending virtual memory address for the program. The beginning virtual memory address and virtual program size define the range of virtual memory addresses at which the program is located.

As shown in step 514, for each Elf file 417 that contains data requiring initialization during fundamental operating system data initialization step 210, microprocessor 101 uses the physical memory address range and the virtual memory address range specified in the program header for each program to initialize the segment table and page table entries corresponding to data to be mapped from ROM 103 to virtual addresses, e.g., read-only data 421, as explained above with respect to step 503 of Figure 5A and incorporated herein by reference. Likewise, in step 515, data to be copied to RAM 102 is copied into RAM 102 and the data mapped from RAM 102 to virtual addresses, as described above with respect to step 506 of Figure 5A and incorporated herein by reference.

As in the embodiment of fundamental operating system data initialization step 210 illustrated in Figure 5A, in a further embodiment of the embodiment of fundamental operating system data initialization step 210 illustrated in Figure 5B, in step 516, for each piece of data copied to RAM 102, the segment table and page table entries are marked to indicate that the corresponding physical memory location has been used. This embodiment is used in recovery boot process 200 illustrated in Figure 2A. In a further embodiment, the segment table and page table entries are not marked, i.e., step 516 is eliminated from the embodiment of fundamental operating system data initialization step 210 illustrated in Figure 5B. This further embodiment is used in recovery boot process 260 (Figure 2B).

After fundamental operating system data initialization step 210, the operating system kernel of computer system 100 can run. Once the operating system kernel can run, the virtual memory location of the region table is accessible. The virtual memory location of the region table is translated, using the segment table and page tables, into a physical memory address so that microprocessor 101 can access the contents of the region table. The region table indicates, among other things, whether a particular virtual memory address or range of virtual memory addresses store persistent data 112. Using these virtual memory addresses, the segment table or page table entries, as appropriate, corresponding to virtual memory addresses of persistent data 112 are, in one embodiment, marked as used, like the fundamental operating system data above, so that the physical memory addresses corresponding to the persistent data 112 can be removed from the system free list during the system free list initialization step 230.

Figure 6 is a diagram of persistent region recovery step 220 according to an embodiment of the invention. First, in step 601, the region table that stores the persistent data 112 attribute for each virtual memory location is located. In one embodiment of the invention, the beginning of the region table is located at a pre-determined virtual memory address, as defined by the operating system, so that the beginning physical memory location of the region table in RAM 102 can be found by traversing the segment table and page tables. In a further embodiment, in which the PenPoint operating system supplied by Go Corp. operates on an AT&T Hobbit microprocessor, the beginning of the region table is located at a virtual address equal to E0600000H.

In another embodiment, the virtual memory address of the beginning of the region table is "discovered" by accessing the boot data structure which is stored at a pre-determined physical memory location in ROM 103 and can be accessed at an early stage of execution of the boot code.

Each region, i.e., a contiguous block of virtual memory, is represented by a region table entry in the region table. Each region table entry includes the range of virtual memory addresses defining the region, permission attributes, and a persistence flag, among other things. A region table entry is retrieved in step 602 and examined in step 603 to determine whether the region corresponding to the region table entry is marked as persistent. Marking of a region as persistent occurs, as explained above, during the first initialization of the operating system of computer system 100 or during normal operation of computer system 100.

If the region is not marked as persistent, processing transfers from step 603 to step 602 and the next region table entry is retrieved. Conversely, if the region is marked as persistent, processing transfers from step 603

to step 604. In step 604, the virtual memory address range of the region is ascertained from the region table entry.

After step 604, in step 605, the virtual memory base address of the first virtual page in the virtual memory address range is ascertained. In step 606, the virtual memory base address of the virtual page is translated to a physical memory address using the segment table and page tables.

After translation of the virtual memory address to the physical memory address, in step 607, the corresponding segment table entry is checked to see if the segment table entry is marked. If the segment table entry is not marked, then, in step 608, the segment table entry is marked. Referring to Figure 3A, the segment table entry, e.g., segment table entry 311-2, is marked by appropriately setting one of the bits of region 313.

Once the segment table entry is either marked (steps 607 and 608) or determined to have been previously marked (step 607), in step 609, the page table entry in the appropriate page table is marked. Referring to Figure 3A, the page table entry, e.g., page table entry 321-4, is marked by appropriately setting one of the bits of region 323. By marking the segment table and page table entries of each virtual page within a region that stores persistent data, the physical memory base address of each physical page storing persistent data 112 is marked so that the physical memory base address can later be removed from the system free list during system free list initialization step 230.

After each page is marked in the page table (step 609), in step 610, a determination is made as to whether the end of the virtual memory address range for the region being processed has been reached. If the end of the range has not been reached, then the virtual memory base address of the next virtual page is determined

(step 605), the virtual memory base address is translated to a physical memory base address (step 606), and the associated segment and page table entries are marked (steps 607, 608 and 609), as described above. If the end of the range has been reached, then the next region table entry is retrieved (step 602) and examined to determine whether the region includes persistent data (step 603). All region table entries are checked in the manner described, all persistent data 112 is identified and the segment table and page table entries (i.e., physical memory addresses) corresponding to persistent data are marked. When persistent region recovery step 220 is complete, the system free list can be initialized.

Figure 7A is a block diagram of system free list initialization step 230 of Figure 2A according to an embodiment of the invention. At any given time during operation of computer system 100, the system free list includes all physical pages of memory that have not yet been allocated, i.e., physical memory locations at which data has not been stored.

During the first initialization of the operating system of computer system 100, a physical memory page is allocated for the system free list and marked as persistent. Thus, during the persistent region recovery step 220 of recovery boot process 200, the physical memory page at which the system free list is stored is recovered. This page is used to store the new system free list generated during recovery boot process 200 by the system free list initialization step. It is necessary to allocate the same physical page for the system free list to ensure that persistent data 112 is not overwritten by allocation of a physical page for the system free list.

To begin system free list initialization step 230, in step 701, all physical pages of memory, including those previously allocated during recovery boot process 200, i.e., pages allocated during fundamental operating system

data initialization step 210 and persistent data recovery step 220, are added to the system free list.

In step 702, the region table is located and accessed. In step 703, the first region table entry is accessed.

As discussed above with respect to Figure 6, each region table entry identifies a range of virtual memory addresses. In step 704, the beginning virtual memory address of each virtual memory page is accessed. In step 705, the segment table and page table entries corresponding to the beginning virtual memory address are checked to see if the entries are marked to indicate that the corresponding physical memory page has been allocated.

If the physical page has been allocated, in step 706, the physical page is removed from the system free list. In step 707, a determination is made as to whether the last virtual page in the region has been checked. If the last virtual page has not been checked, then the next virtual page in the region is accessed (step 704) and checked to see if the corresponding physical page has been allocated. If the last virtual page in the region has been checked, then, in step 708, a determination is made as to whether the region table entry being processed is the last region table entry. If the region table entry is the last region table entry, then the recovery boot process continues. If the region table entry is not the last region table entry, then the next region table entry is accessed (step 703) as described above.

If, at step 705, it is determined that the physical page has not been allocated, then the physical page is not removed from the system free list. A determination is made as to whether the last page in the region has been reached (step 707) and, if appropriate, whether the region table entry being processed is the last region table entry (step 708), and processing of region table entries and virtual pages continues as described above. Each region

is checked for allocated pages, and all allocated pages are removed from the system free list.

Figure 7B is a block diagram of system free list initialization step 231 of Figure 2B according to an embodiment of the invention. In steps 701, 702 and 703, every physical memory page is added to system free list, and the first region table entry of the region table is accessed.

In step 711, each region table entry is checked, by checking the persistent attribute of the region table entry, to determine if the persistent data is stored at the physical memory locations corresponding to the region. If persistent data is not stored, the next region table entry is accessed (step 703). If persistent data is stored, in step 712, the beginning virtual memory address of each virtual memory page in the region is translated to the beginning physical memory address of the corresponding physical memory page, and the physical memory page is removed from the system free list.

In step 708, a determination is made as to whether the region table entry is the last region table entry. If not, then the next region table entry is accessed (step 703) and checked for the presence of persistent data (step 704). If so, then, in step 709, the recovery boot process continues.

Since the region table includes only virtual addresses that have been allocated during previous operation of computer system 100, scanning the region table eliminates checking of virtual addresses with which no data is associated. However, in another embodiment of the invention, all segment and page table entries are checked to determine if the corresponding physical page has been allocated, and all allocated pages are removed from the system free list.

When the last region has been examined, system free list initialization step 230 is complete. The remaining

physical memory pages in the free list - those that have not been removed during system free list initialization step 230 - have not been allocated and are free to be used to store any data without possibility of destroying persistent data 112. This is because, up to the point of system free list initialization step 230, no physical memory pages were allocated without making the allocation based upon the virtual memory map, i.e., the segment table and page table entries, existing prior to the system reset step 205. In other words, fundamental operating system data initialization step 210 and persistent region recovery step 220 lock out physical memory locations from the system free list so that these locations cannot later be used to store other data that would destroy the fundamental operating system data or, in particular, the persistent data 112. Once system free list initialization step 230 is complete, the remainder of recovery boot process 200 can proceed without fear of destroying persistent data 112.

After system free list initialization step 230, system start-up step 240 is performed, as discussed in more detail above with respect to Figure 2. Once system start-up 240 is complete, file system recovery step 250 takes place.

The file system is a set of data structures that describe a name-space and data that's contained within that name-space. The file system includes information for each of the files such as the state of the file, whether the file is opened or closed, whether the file has been written to, permission attributes, and whether the file is write-protected or not.

The first time that the file system is initialized, a heap is created. A heap pointer is stored in the boot data structure. The file system is initialized from ROM 103 by scanning all ROM items 413 in ROM 103 and marking all items that have a file name. Elements in the

heap are used to record file system structures for potentially allocating new heap entities and for pointing back into the ROM so that it looks like the file system is composed only of a set of ROM files.

Over time, new files are allocated or deleted. New files are allocated either from the heap, or directly from a memory allocator that is part of computer system 100 and the files referenced within the heap. RAM files are marked persistent when they are allocated. Files which were initialized from ROM 103 can be deleted or modified by copying the file into RAM 102 and then putting the file back into the file system, i.e., using the same mechanism as creating a file originally in RAM. Consequently, the physical memory locations allocated for these "new" files are also marked as persistent data 112.

In file system recovery step 250, the file system is recovered by retrieving the heap pointer from the boot data so that microprocessor 101 can locate files in the file system once again. After file system recovery step 250, recovery boot process continues by initializing any remaining operating system data (not shown).

Various embodiments of the invention have been described. The descriptions are intended to be illustrative, not limitative. Thus, it will be apparent to one skilled in the art that certain modifications may be made to the invention as described without departing from the scope of the claims set out below.

We claim:

1. In a computer system including a microprocessor and a memory storing persistent data, a method for recovery contents of the memory, after a system reset, to a state that allows user control of the operation of the microprocessor, the method comprising the steps of:

recovering a portion of fundamental operating system data such that said portion of fundamental operating system data is stored within the memory in locations different from locations that stored persistent data prior to the recovery; and

recovering the persistent data.

2. A method as in Claim 1, wherein:

each location in the memory at which data is stored is identified by a virtual memory address; and

the step of recovering fundamental operating system data further comprises:

retaining, after the system reset, the virtual memory address, existent before the system reset, of each of the first set of fundamental operating system data;

translating the virtual memory address of each of the first set of fundamental operating system data to a physical memory address using translation data; and

storing each of the fundamental operating system data at a location in the memory corresponding to the physical memory address determined during the step of translating.

3. A method as in Claim 2, wherein the step of recovering the fundamental operating system data further comprises marking each of the translation data to indicate that locations in the memory that store the first set of

fundamental operating system data cannot be used to store other data.

4. A method as in Claim 2, wherein:

persistent data is identified, prior to the system reset, by a tag associated with the virtual address of each of the persistent data; and

the step of recovering persistent data comprises:

identifying virtual memory addresses that have been tagged;

translating each of the tagged virtual memory addresses to a physical memory address using translation data; and

recovering data stored at the locations in the memory corresponding to the physical memory addresses such that other data cannot be stored, subsequent to the step of recovering persistent data, in the locations in the memory at which persistent data are stored.

5. A method as in Claim 4, wherein:

the step of recovering the fundamental operating system data further comprises marking the translation data to indicate that locations in the memory that store the first set of fundamental operating system data cannot be used to store other data; and

the step of recovering persistent data further comprises marking the translation data to indicate that locations in the memory that store persistent data cannot be used to store other data.

6. A method as in Claim 5, further comprising the step of constructing a list of physical memory addresses corresponding to locations in the memory which can be used to store data during the recovery subsequent to the step

of constructing, the list including all physical memory addresses corresponding to locations in the memory except physical memory addresses corresponding to the marked translation data.

7. A method as in Claim 1, wherein:

each location in the memory at which data is stored is identified by a virtual memory address;

persistent data is identified, prior to the system reset, by a tag associated with the virtual address of each of the persistent data; and

the step of recovering persistent data comprises:

identifying virtual memory addresses that have been tagged;

translating each of the tagged virtual memory addresses to a physical memory address using translation data; and

recovering data stored at the locations in the memory corresppnding to the physical memory addresses such that other data cannot be stored, subsequent to the step of recovering persistent data, in the locations in the memory at which persistent data are stored.

8. A method as in Claim 7, wherein the step of recovering persistent data further comprises marking the translation data to indicate that locations in the memory that store persistent data cannot be used to store other data.

9. A method as in Claim 1, further comprising the step of constructing a list of physical memory addresses corresponding to locations in the memory which can be used to store data during the recovery subsequent to the step of constructing, wherein:

physical memory addresses corresponding to locations in the memory at which the first set of fundamental operating system data are stored are excluded from the list; and

physical memory addresses corresponding to locations in the memory at which the persistent data are stored are excluded from the list.

10. In a computer system including a microprocessor and a memory storing persistent data, a method for recovery contents of the memory, after a system reset, to a state that allows user control of the operation of the microprocessor, the method comprising the steps of:

recovering fundamental operating system data such that a first set of fundamental operating system data is stored within the memory and such that none of the first set of fundamental operating system data is stored in locations in the memory that stored persistent data prior to the system reset; and

constructing a list of physical memory addresses for locations in the memory which can be used to store data during the recovery subsequent to the step of constructing, wherein:

physical memory addresses for locations in the memory at which the first set of fundamental operating system data are stored are excluded from the list; and

physical memory addresses for locations in the memory at which the persistent data are stored are excluded from the list.

11. A method as in Claim 10, wherein:

each location in the memory at which data is stored is identified by a virtual memory address; and

the step of recovering fundamental operating system data further comprises:

retaining, after the system reset, the virtual memory address, existent before the system reset, of each of the first set of fundamental operating system data;

translating the virtual memory address of each of the first set of fundamental operating system data to a physical memory address using translation data; and

storing each of the first set of fundamental operating system data at a location in the memory corresponding to the physical memory address determined during the step of translating.

12. A method as in Claim 11, wherein the step of recovering the fundamental operating system data further comprises marking each of the translation data to indicate that locations in the memory that store the first set of fundamental operating system data cannot be used to store other data.

13. A method as in Claim 12, wherein the step of constructing further comprises including all physical memory addresses corresponding to locations in the memory except physical memory addresses corresponding to the marked translation data.

14. In a computer system including a microprocessor and a memory storing persistent data, a method for recovery contents of the memory, after a system reset, to a state that allows user control of the operation of the microprocessor, the method comprising the steps of:

prior to the system reset, marking locations in the memory that store persistent data; and

after the system reset, recovering the persistent data.

15. A method as in Claim 14, wherein:

each location in the memory at which data is stored is identified by a virtual memory address;

the step of marking locations in the memory further comprises associating a tag with the virtual memory address corresponding to each location in the memory at which persistent data is stored; and

the step of recovering persistent data further comprises:

identifying virtual memory addresses that have been tagged;

translating the tagged virtual memory addresses to physical memory addresses using translation data; and

recovering data stored at the locations in the memory corresponding to the physical memory addresses such that data cannot be stored, subsequent to the step of recovering persistent data, in the locations in the memory that store persistent data.

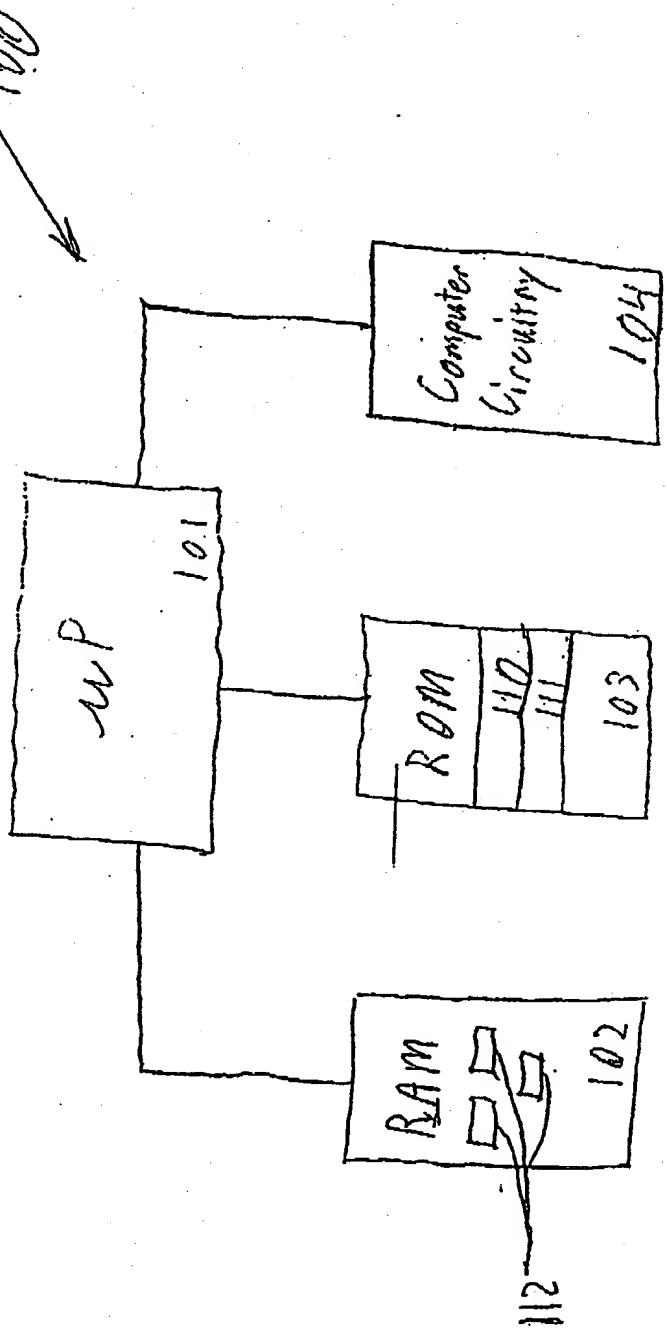


Figure 1

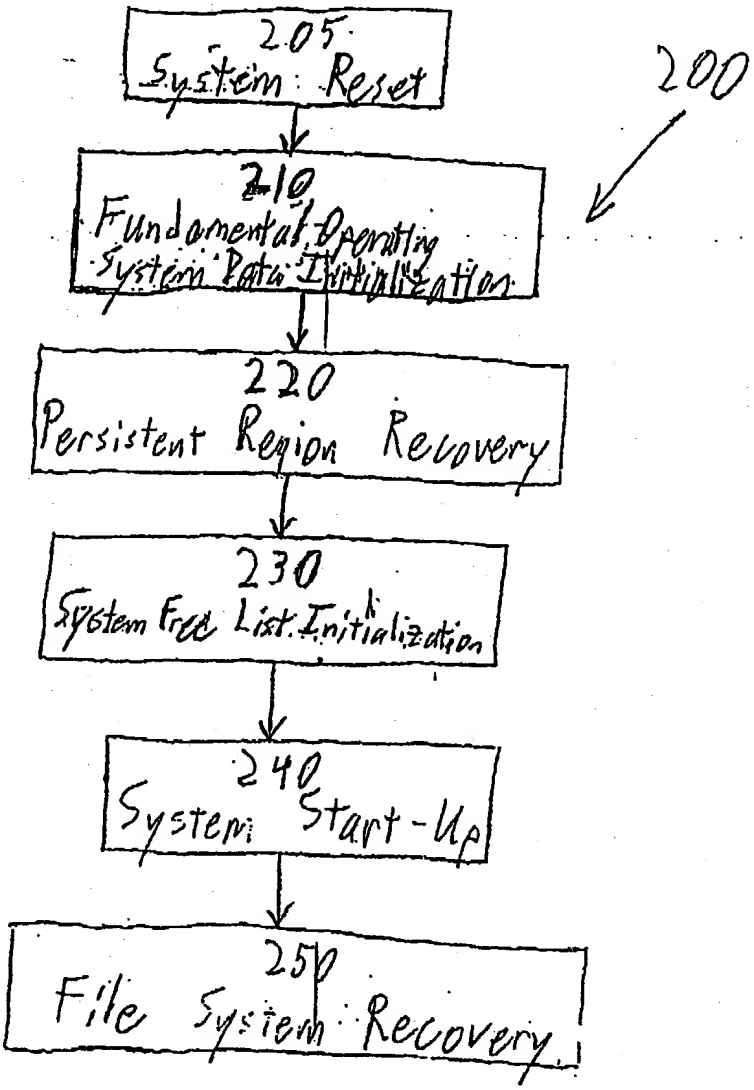

205
System Reset
200
210
Fundamental Operating System Data Initialization
220
Persistent Region Recovery
230
System Free List Initialization
240
System Start-Up
250
File System Recovery

Figure 2A

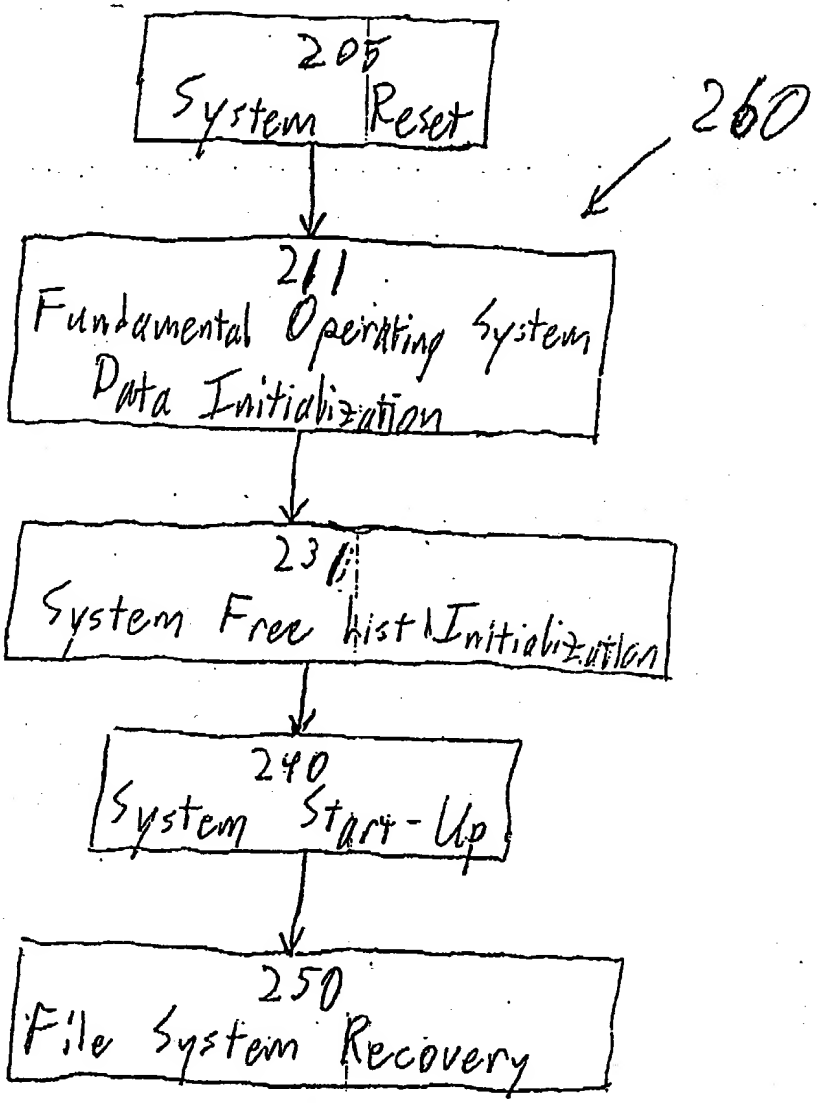

205
System Reset
260
211
Fundamental Operating System Data Initialization
230
System Free List Initialization
240
System Start-Up
250
File System Recovery

Figure 2B.

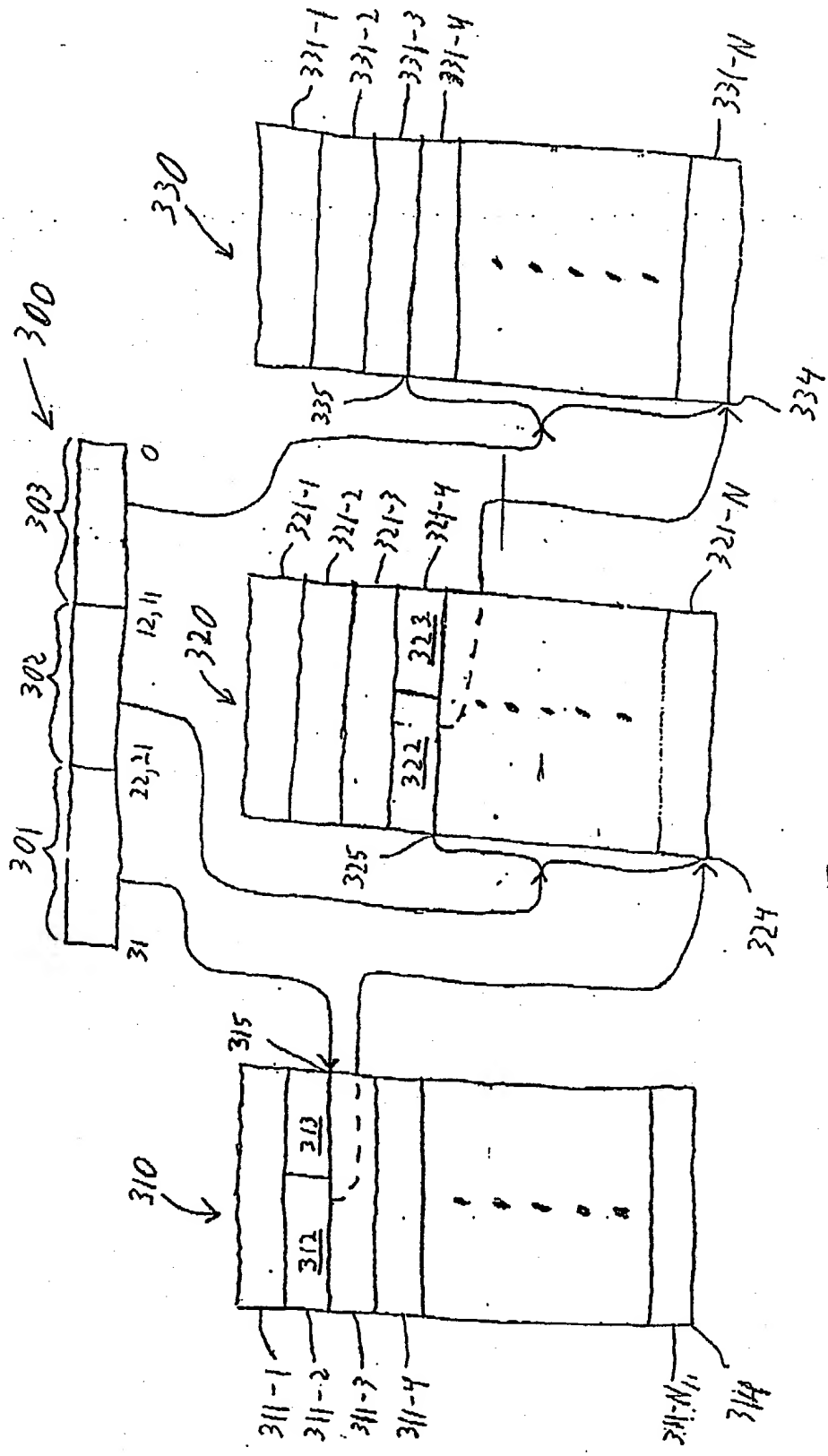

300
301
302
303
31
0
12,11
22,21
310
311-1
311-2
311-3
311-4
311-N1
314
312
313
315
320
321-1
321-2
321-3
321-4
321-N
322
323
324
325
330
331-1
331-2
331-3
331-4
331-N
334
335

Figure 3A

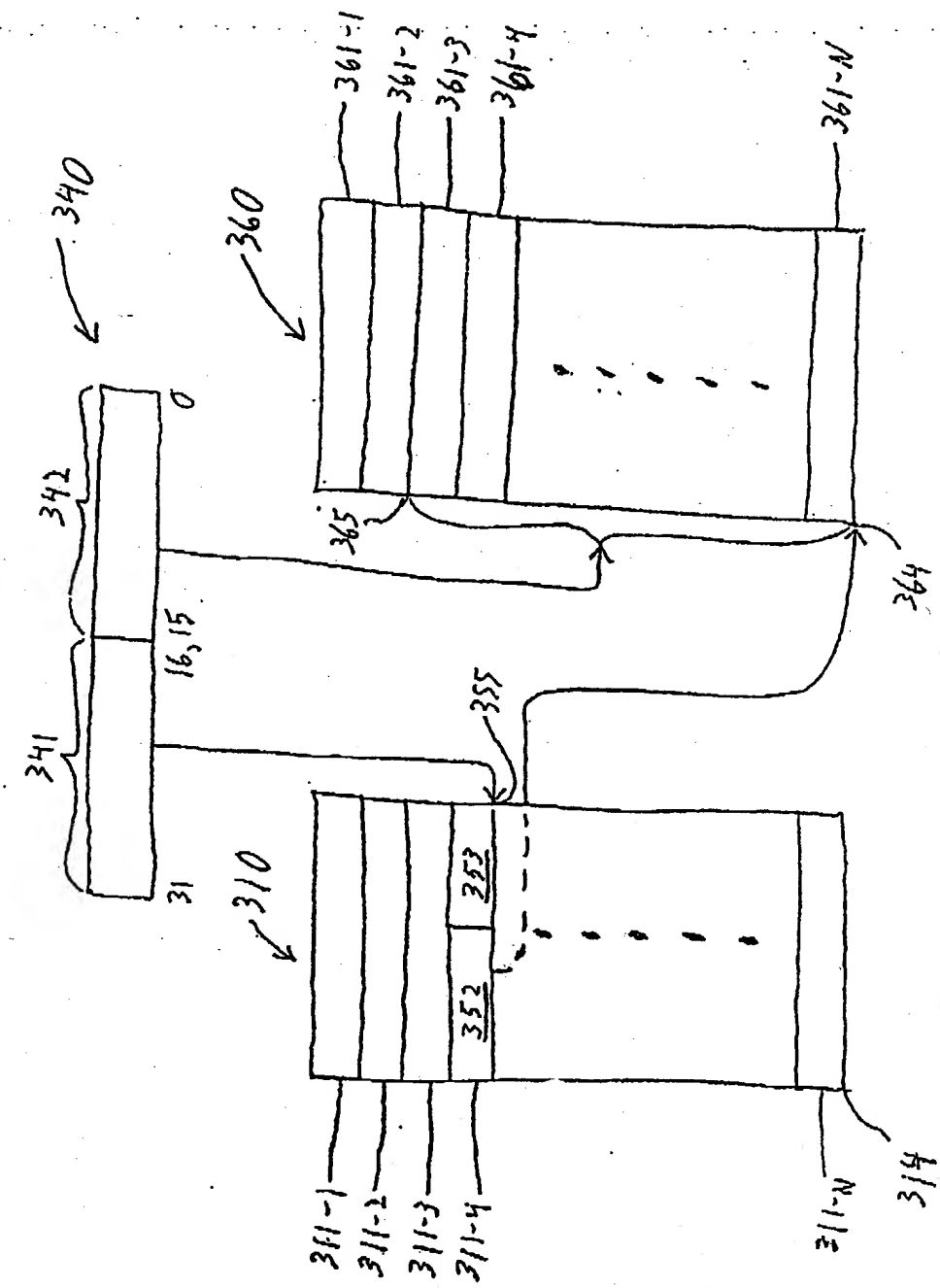

340
360
361-1
361-2
361-3
361-4
361-N
342
0
365
344
16,15
341
355
31
310
253
352
311-1
311-2
311-3
311-4
311-N
314

Figure 3B

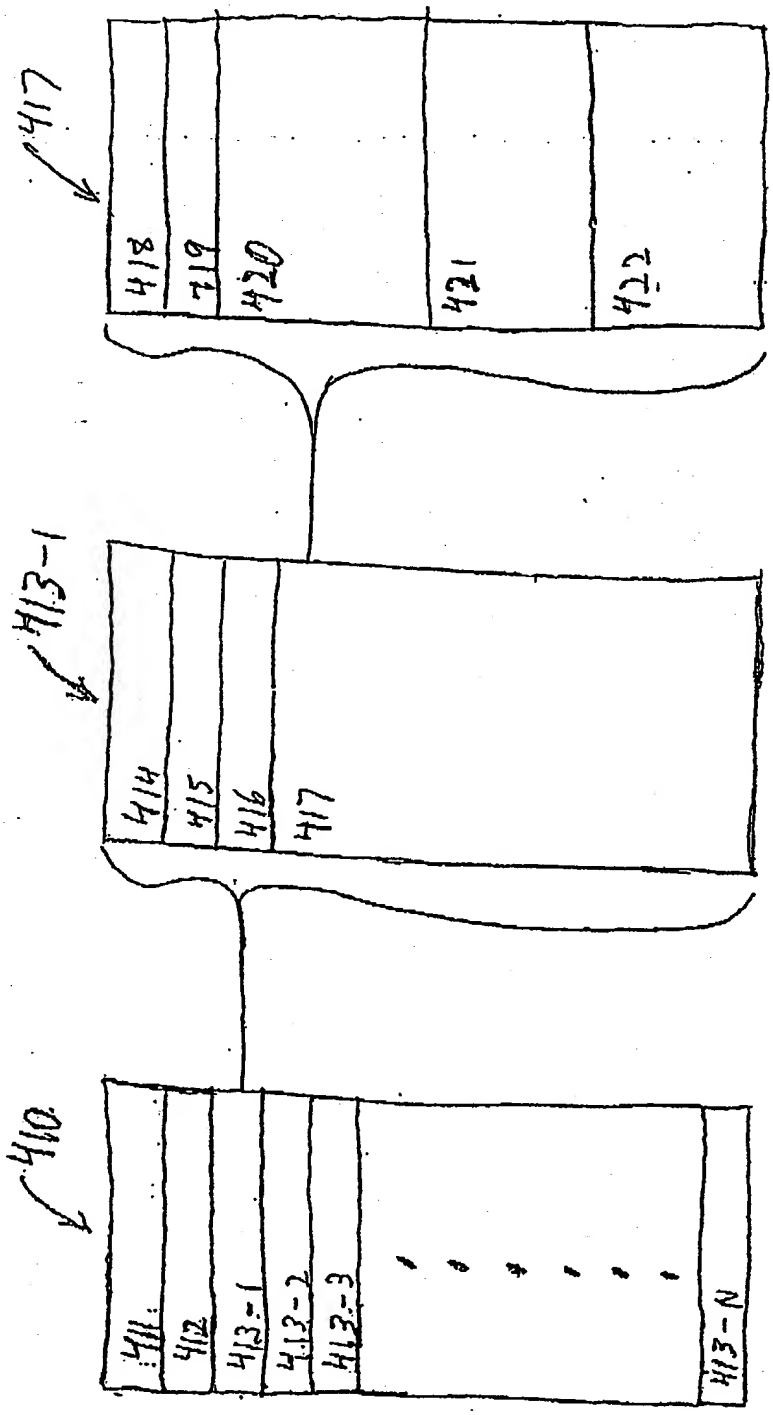


Figure 4A

Figure 4B

Figure 4C

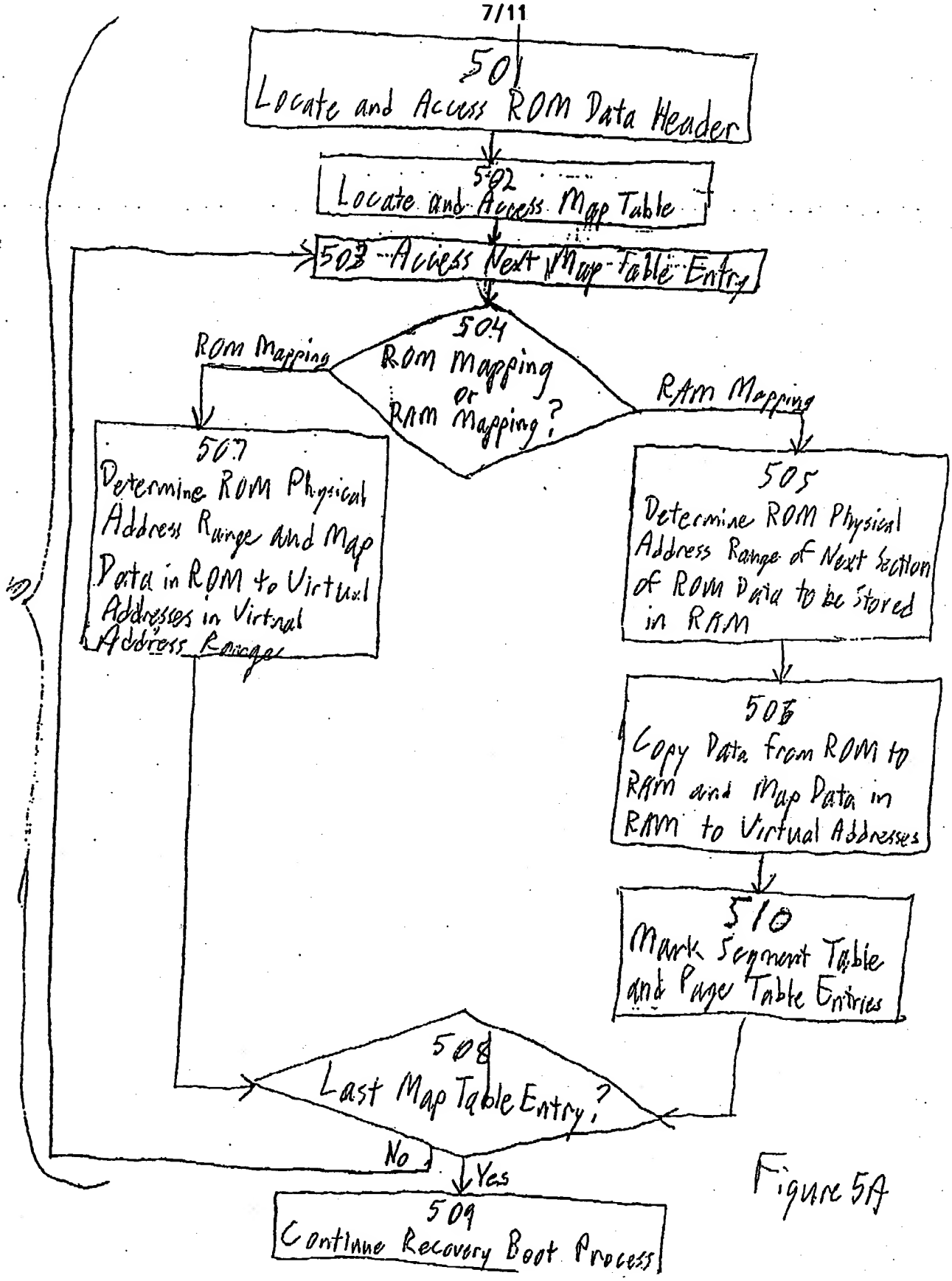

501
Locate and Access ROM Data Header
502
Locate and Access Map Table
503
Access Next Map Table Entry
504
ROM Mapping or RAM Mapping?
ROM Mapping
RAM Mapping
507
Determine ROM Physical Address Range and Map Data in ROM to Virtual Addresses in Virtual Address Range
505
Determine ROM Physical Address Range of Next Section of ROM Data to be Stored in RAM
506
Copy Data from ROM to RAM and Map Data in RAM to Virtual Addresses
510
Mark Segment Table and Page Table Entries
508
Last Map Table Entry?
No
Yes
509
Continue Recovery Boot Process

Figure 5A

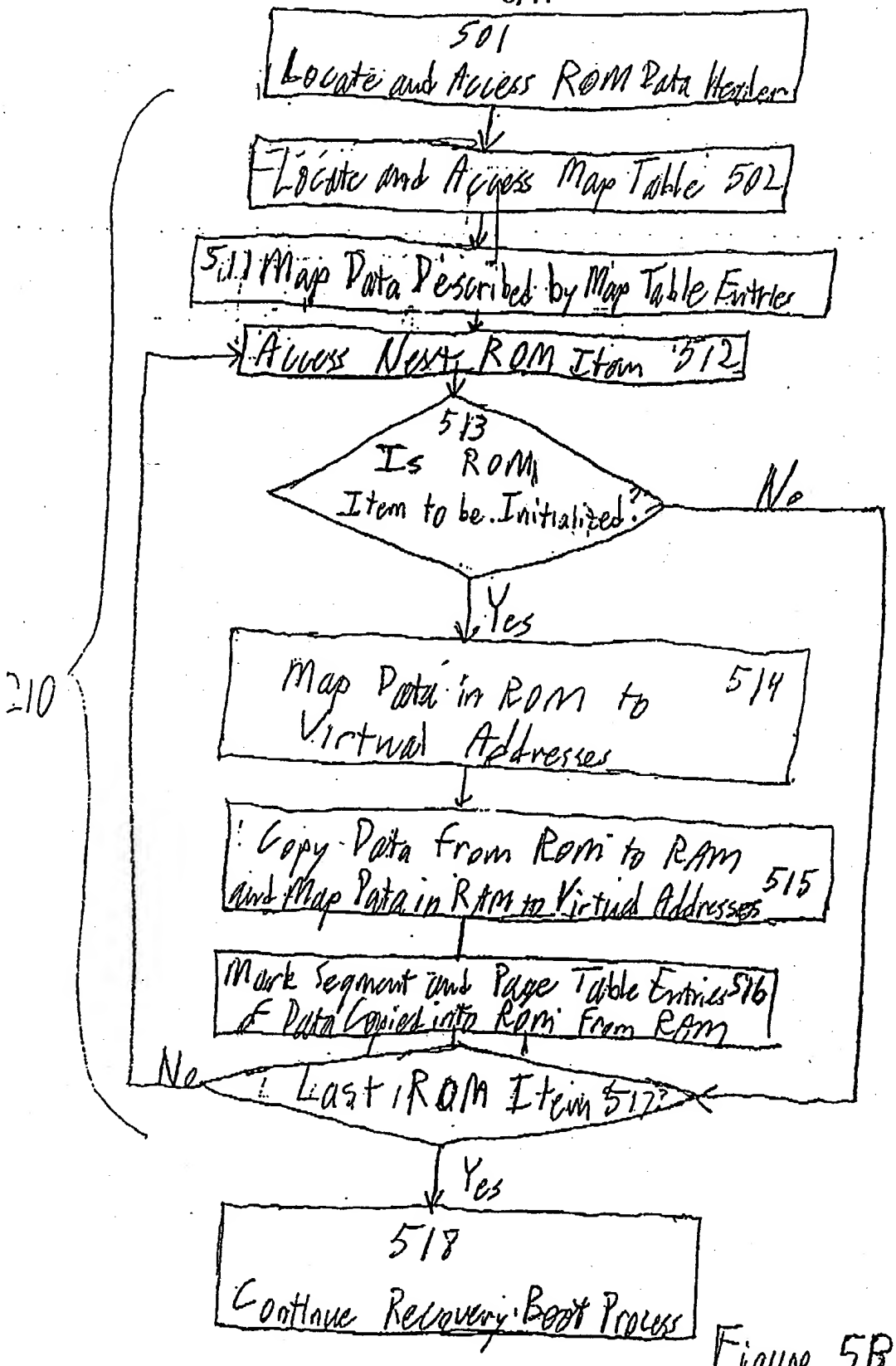

501
Locate and Access ROM Data Header
Locate and Access Map Table 502
511 Map Data Described by Map Table Entries
Access Next ROM Item 512
513
Is ROM Item to be Initialized?
No
Yes
210
Map Data in ROM to Virtual Addresses 514
Copy Data from Rom to RAM and Map Data in RAM to Virtual Addresses 515
Mark Segment and Page Table Entries of Data Copied into Ram From RAM 516
No
Last ROM Item 517?
Yes
518
Continue Recovery Boot Process

Figure 5B

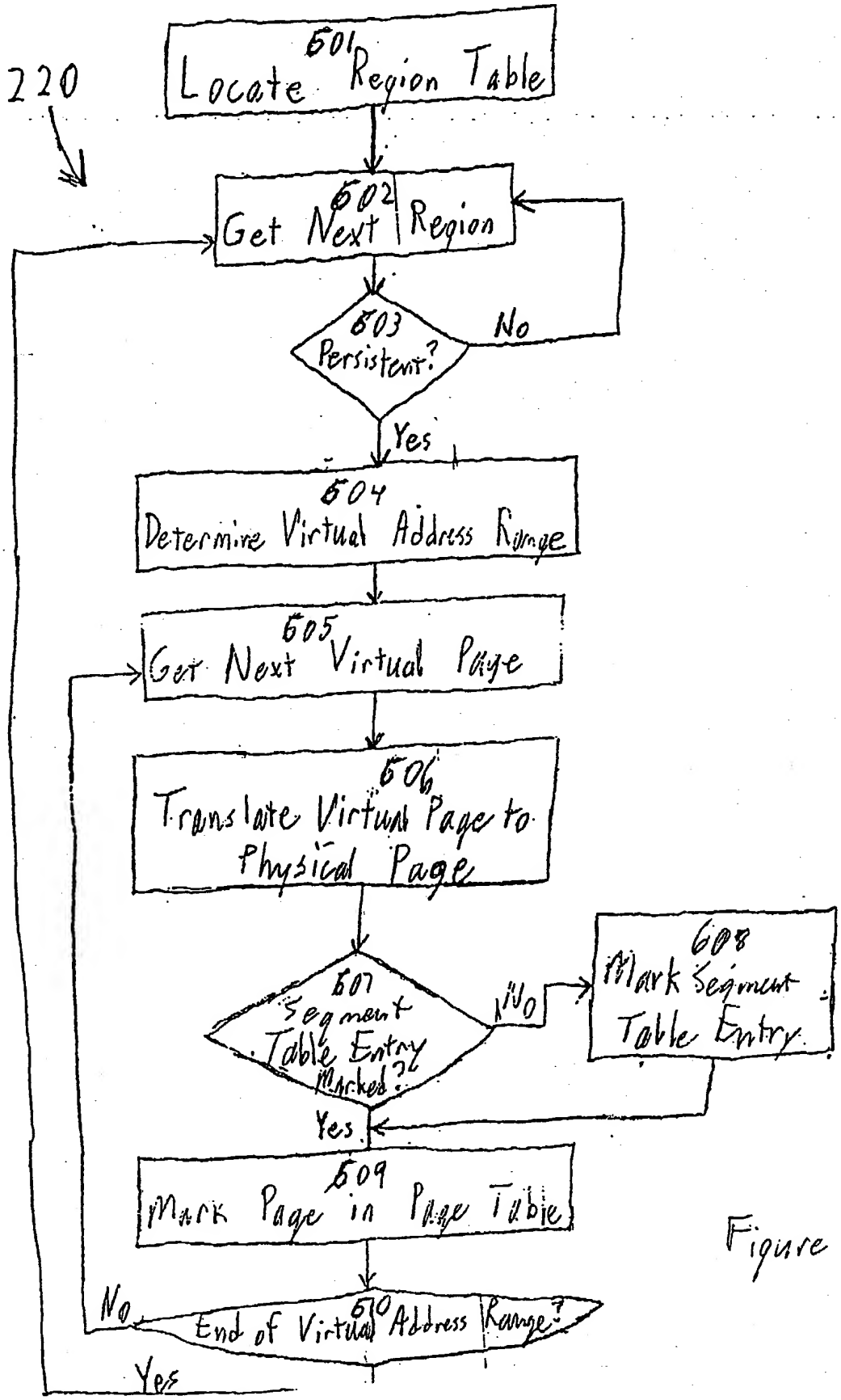

220
601 Locate Region Table
602 Get Next Region
603 Persistent?
No
Yes
604 Determine Virtual Address Range
605 Get Next Virtual Page
606 Translate Virtual Page to Physical Page
607 Segment Table Entry Marked?
No
608 Mark Segment Table Entry
Yes
609 Mark Page in Page Table
610 End of Virtual Address Range?
No
Yes

Figure 6

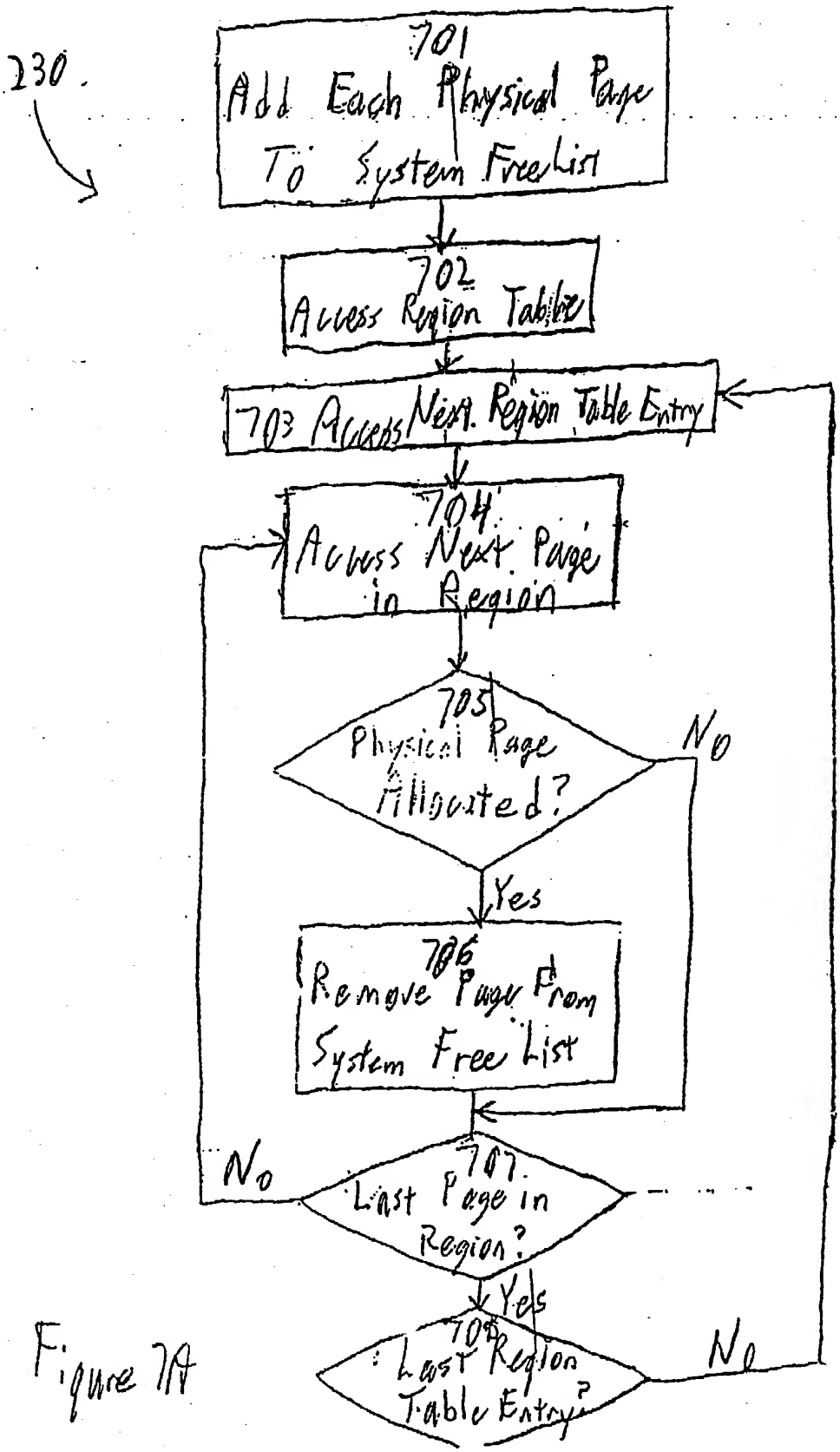

230
701
Add Each Physical Page To System Free List
702
Access Region Table
703 Access Next Region Table Entry
704
Access Next Page in Region
705
Physical Page Allocated?
No
Yes
706
Remove Page From System Free List
707
Last Page in Region?
No
Yes
708
Last Region Table Entry?
No

Figure 7A

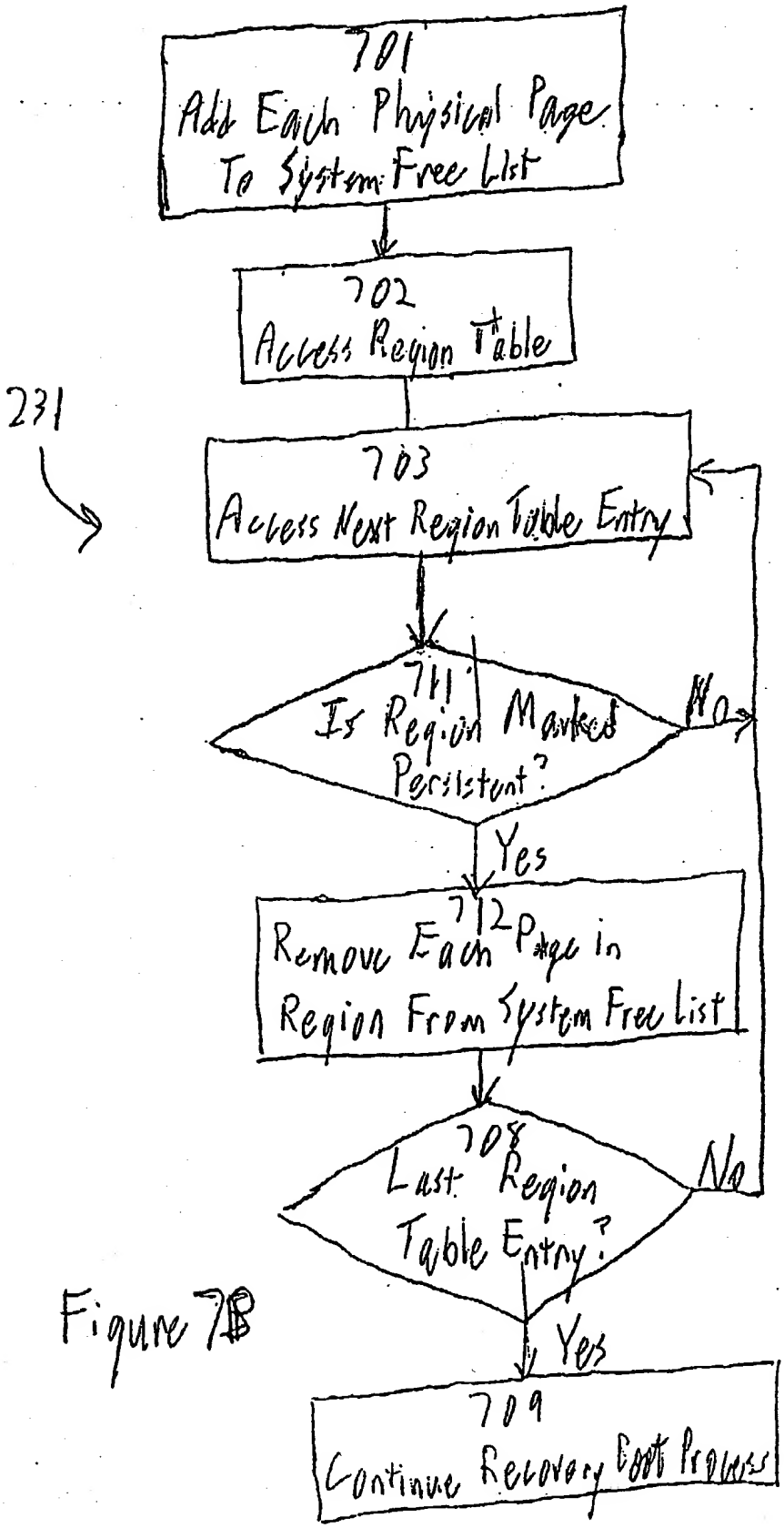

701
Add Each Physical Page To System Free List
702
Access Region Table
231
703
Access Next Region Table Entry
711
Is Region Marked Persistent?
No
Yes
712
Remove Each Page in Region From System Free List
708
Last Region Table Entry?
No
Yes
709
Continue Recovery Boot Process

Figure 7B

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/12567

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 11/00
US CL : 395/575
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/575; 371/7, 12; 364/260.5, 260.8

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

PROQUEST, IEEE TRANSACTIONS 1988-PRESENT

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

ORBIT, APS, PROQUEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,247,669 (ABRAHAM ET AL) 21 September 1993, abstract, col. 6, lines 38-54, figure 4	1, 10, 14
Y, P	US, A, 5,291,593 (ABRAHAM ET AL) 01 March 1994, abstract, col. 1, lines 50-52 , col. 2, lines 21-58	1, 10, 14
Y	US, A, 5,155,856 (BOCK ET AL) 13 October 1992, abstract, col. 1, lines 12-54, etseq	1, 10,14
Y	US, A, 4,680,700 (HESTER ET AL) 14 July 1987, col. 9, lines 19-27	2-9, 11-13, 15
Y	US, A, 4,638,426 (CHANG ET AL) 20 January 1987, see abstract.	2-9, 11-13,1 5

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
05 JANUARY 1995

Date of mailing of the international search report
12APR1995

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
ALBERT DECADY
Telephone No. (703) 305-9600

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/12567

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, P	US, A, 5,333,303 (MOHAN) 26 July 1994, abstract, col. 4, lines 34-59 (passim)	1-15
A	US, A, 5,155,844 (CHENG ET AL) 13 October 1992, abstract	1-15
A, P	US, A, 5,317,752 (JEWETT ET AL) 31 May 1994, abstract	1-15
A	US, A, 5,255,367 (BRUCKERT ET AL) 19 October 1993, entire document	1-15
A	IEEE, 1992 , JEFF CHASE ET AL., "USING VIRTUAL ADDRESSES AS OBJECT REFERENCES", PAGES 245-248	1-15